TECHNICAL REPORT 1884
April 2002

Detecting Failure within Distributed Environments

J. Drummond
SSC San Diego

D. Wells
M. Rahman
The Open Group Research Institute

SSC San Diego
San Diego, CA 92152-5001

SSC SAN DIEGO
San Diego, California 92152-5001

P. A. Miller, CAPT, USN
Commanding Officer

R. C. Kolb
Executive Director

ADMINISTRATIVE INFORMATION

The work described in this report was performed for the Defense Advanced Research Projects Agency (DARPA) Information Technology Office (ITO) by the Advanced Concepts and Engineering Division (Code 241), SSC San Diego; Teknowledge Corporation, Palo Alto, CA; The Open Group Research Institute, Woburn, MA; and System Technology Development Corporation, Herdon, VA.

ACKNOWLEDGMENT

This report was compiled from research and experiments performed within the Quorum Integration, Testbed and Exploitation (QUITE) project. This DARPA ITO Quorum project effort is structured as follows: DARPA ITO, Sponsor; SSC San Diego, Technical/Contracting Lead; Teknowledge Corporation, Integration Lead; System/Technology Development Corporation (S/TDC) and the Open Group (TOG); Integration.

The individuals who directly contributed to this specific effort included Gary Koob, Quorum Program Manager (DARPA ITO); John Drummond and Al Sandlin (SSC San Diego); Neil Jacobstein, Adam Pease, Lou Coker, Jeff Vu, Joe Marclino, and Jim Reynolds (Teknowledge); Mustafizur Rahman and Jim Carroll (TOG); and Arthur Robinson, Manoj Srivastava, Amarendranath Vadlamudi, and Shivakumar Patil (S/TDC).

Active participation by all the QUITE project personnel included extensive discussion regarding the research and experiments conducted within the QUITE project. This document has drawn upon the research and the numerous quality of service management experiment design planning meetings and other related informal writings developed throughout the QUITE project effort.

SB

EXECUTIVE SUMMARY

OBJECTIVE

The main objective of this research effort is initial creation of a viable approach for detecting node failures within resource-managed distributed environments. Current failure detection approaches do not report these failure events promptly and the false detection rates of node failures are often exceedingly high. Therefore, the fundamental object of this research was the production of a fast failure-detection program that allows rapid and accurate detection of node failures within the resource-managed distributed environment.

METHOD

The method for this research effort was multifaceted and included many elements. The initial design for detecting node failures required the development of hypotheses and an experiment. The experiment provided an accurate evaluation of the devised fast failure-detection approach. The development of software and integration of related system programs and the implementation of a testbed environment provided the structure for this work.

CONCLUSION

The developed failure detection system was successfully implemented within the Quorum Integration, Testbed and Exploitation (QUITE) project environments at multiple testbeds. The precise experiments were also created and performed upon failure-detection systems at these multiple test beds. The data resulting from these experiments support the initial hypotheses regarding node failure detection within a resource-managed distributed environment. This report includes the resulting data and conclusions. The overall findings show that the fast failure-detection system was successful in rapidly detecting node failures while maintaining reasonable false failure-detection totals.

CONTENTS

Figures

INTRODUCTION

The experiment described in this report investigated a proposed method for improving the behavior of systems that must meet simultaneous quality of service (QoS) requirements for availability and timeliness. This project addressed improving timeliness during the process of failure recovery for systems based on a virtual synchrony communication method.

Although this problem is exhibited in many deployed systems, the motivation for the details of this experiment was derived from the needs of the HiPer-D testbed at Naval Surface Weapons Center (NAWS), particularly in an antiaircraft warfare (AAW) application. This application, which has a "sensor-to-shooter" time constraint, supports scaleable fault tolerance based on virtual synchrony. When one process fails, other processes detect that failure, reassign tasks as appropriate, and continue operation. Group communication simplifies recovery from various types of failure by ensuring that all remaining participants share a common view of system status.

Using virtual synchrony to realize the common system view, however, introduces a delay whenever the group membership changes. This delay, which persists until the new group reaches consensus on its own membership, leads to a significant impairment in system performance during failure recovery.

A superficial, but educated, analysis of system activities during failure recovery showed that the primary factor leading to the slow recovery was the time required to find and isolate a failed node. A mechanism that could speed up the process of finding a failed node could benefit overall system performance. Analysis of the failure detection mechanisms showed that lack of speed was largely due to indeterminism in the underlying operating system and network infrastructure. A Quorum component that could allocate reserved resources to particular activities could improve the performance of real-time, fault-tolerant systems such as the HiPer-D AAW application.

This experiment investigated whether Carnegie Melon University's (CMU) resource kernel, or a similar component, such as the Oregon Graduate Institute's (OGI's) QUAlity Specification and Adaptive Resource (QUASAR) management for distributed systems could reduce the failure recovery time in systems using group communication for synchronization, as implemented in Cornell's Ensemble[1] toolkit. Positive results would lead to improvements in the performance of NSWC's HiPer-D system. The modifications to the Quorum components and the components developed as part of this experiment would also substantially improve the capabilities of the Quorum Integration, Testbed and Exploitation (QUITE) tool-kit relative to their use in real-time, fault-tolerant systems.

RELEVANT ISSUES

The top-level goal of the experiment was to determine whether the CMU resource kernel could reduce the time to recover from node failures in applications using Ensemble's group communication facilities. CPU was the only resource managed in this experiment. The network was configured so that its performance was not a bottleneck in the experiment.

---

[1] The description of group communication provided here is greatly oversimplified and many details irrelevant to this experiment are omitted. Heartbeat protocols include design aspects to alleviate some of the drawbacks described here. These design aspects did not affect the conclusions derived from this experiment.

The experiment attempted to validate the assumption that failure detection plays a major role in determining the speed of failure recovery as viewed by a real-time, fault-tolerant application.

Finally, the experiment developed insights into the application of the newly developed fast failure-detection component. Real-time, fault-tolerant applications are highly engineered, and no system designer could be expected to arbitrarily use a single component based only on a single measurement. Instead, each placement of a component was evaluated in its operating context. Thus, characterization of the fast failure detector was fundamental to its evaluation.

TECHNICAL BACKGROUND

Group communication in fault-tolerant systems such as HiPer-D detects failures and reports those failures to the application, which then begins a recovery process. The recovery process must reallocate the processing tasks among the remaining nodes. Although recovering from failures is essential in a high-availability application, the recovery process is expensive in time and consumable resources. The group communication implementation of the virtual synchrony communication model is not a good choice for systems with many failures. In many real systems, failures are rare and the cost of group communication failure recovery is justified. A problem arises, however, if the system generates excess false positives. (A false positive in this analysis is the case where the failure detector function shows that a remote node has failed, but the remote node is still operational.) Applications must respond to false positives as though they were actual node failures, and the resulting system reconfiguration degrades system performance during otherwise normal operation.

The crux of this experiment and of the fast failure-detection component is that failure detection in many systems, including Ensemble, is largely separable and implemented as a heartbeat function. The heartbeat function consists of two parts: a heartbeat generator, which sends messages periodically, and a census taker, which checks that messages have been received from all group members. If the census taker does not receive a message from a group member within the required time, it assumes that the group member has failed and notifies the membership manager, which ejects that group member, and initiates the reformation of a group comprising the remaining group members.

The heartbeat timeout period is the overarching determinant of the failure detection period and, therefore, of the application recovery time. Thus, the obvious method for reducing the failure detection time was to reduce the heartbeat timeout period. Unfortunately, the arrival of heartbeat messages is stochastic, and reducing the timeout period increases the rate of false positives, which leads to reduced system effectiveness. The result is a minimum practical value for the timeout period, which is dependent upon the operational context of the entire system, particularly the load imposed by the application.

Many factors cause the non-determinism of message delivery, including the operating system scheduling, threading within the communication protocol stack, shared resources within the communication protocol stack, false positive elimination strategy within the heartbeat and membership functions, network communication strategy, hardware, radio frequency (RF) environmental noise, and software error correction strategies. A primary contributor in practical systems is congestion of shared resources. Shared hardware elements such as the CPU or network buffers need a non-zero time period before reassignment. Each step in message processing introduces more variability to the message delivery time. A distribution curve characterizes the delivery time. This curve is normally drawn with delivery time on the horizontal axis and the number

of messages with that delivery time along the vertical axis. The shape of the distribution curve is zero messages for small values of time, then a high peak, then a downward sloping curve that trails off toward zero (Figures 1 and 2). Placing a timeout on heartbeat messages can be viewed as placing a vertical rejection line on the distribution curve at some delivery time, which is not equal to the selected timeout. Message deliveries to the left of the line lead to normal operation. Message deliveries to the right lead to false failure detection and ejection of group members.

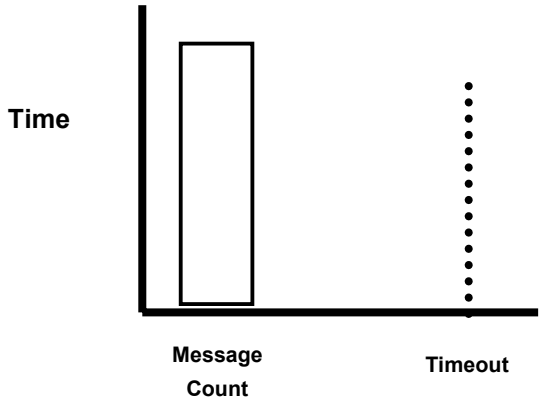

Figure 1. Ideal message delivery distribution.

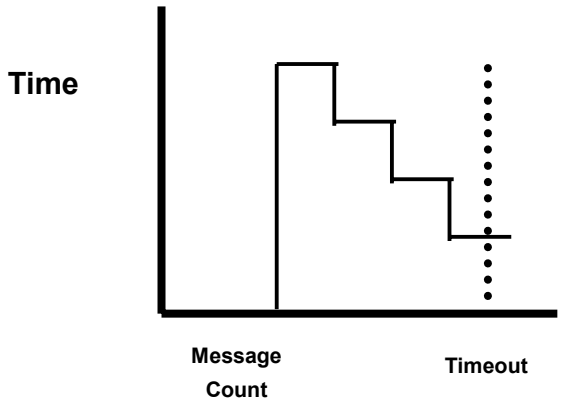

Figure 2. Actual message delivery distribution.

Contention for shared resources increases the probability that the delivery time for any particular message will exceed the timeout period. Reservation of resources for exclusive use in the message delivery should adjust the message delivery distribution curve closer to the ideal (thus increasing the sharpness of the peak) and reduce that probability. A corollary is that the timeout period could be reduced while maintaining the same probability of false positives.

The heartbeat messages are particularly critical in determining the minimum timeout period. Normal application messages are part of an end-to-end sequence. If one message between two application components exceeds the timeout, the overwhelming odds are that the delay will be counteracted by shorter delivery times on other messages in other parts of the application path.

Heartbeat messages, however, are more critical to the determination of node failure. First, to allow the overall application path to recover within its time constraints, the heartbeat timeout period must be significantly smaller than the end-to-end time constraint, perhaps by an order of magnitude.

Second, the exchange of heartbeat messages must be maintained even when there is no application message activity. Consider an application involving 10 nodes in a group with a 1-second timeout and a probability that only one message in a thousand will be delivered late. Each node must receive a message from every other node. A hundred messages (10 nodes squared) are exchanged each second. Such a system will show one false positive (and thus a system reconfiguration) every 10 seconds, probably a nonviable configuration.

Given that indeterminacy of CPU assignment is the relevant problem in addressing timeliness in fault-tolerant applications, one solution might be to use a commercial off-the-shelf (COTS) real-time operating system such as LynxOS or Solaris 2, and assign priorities to individual tasks and/or threads within the application. This approach is unattractive because the isolation of related components and assignment of priorities is tedious and error-prone. Use of COTS products or pseudo-COTS components such as Ensemble introduce black box elements. No interfaces allow the application to assign priorities to internal processing within those components. In many cases, the application is not even aware that additional threads exist.

The design of this experiment focused upon the failure detector. The failure detection function was isolated into a separate execution unit, which could then be provisioned with reserved resources. The use of a separate component also eliminated most anomalies due to contention for shared resources such as memory buffers. The end-result should be a "sharpened" message delivery distribution curve.

EXPERIMENT ENVIRONMENT

The experiment environment was constructed from the Defense Advanced Research Projects Agency (DARPA) Quorum Integration Test & Exploitation project (QUITE) testbed. The QUITE testbeds were located at multiple locations, including the Open Group Laboratory, Woburn, MA; the Teknowledge Laboratory, Palo Alto, CA; the System/Technology Development Corporation Laboratory, Herndon, VA; and the SPAWAR Systems Center, San Diego (SSC San Diego) Laboratory, San Diego, CA. The QUITE testbeds used numerous heterogeneous software systems and various operating systems, including Linux® and Windows NT®. The multiple QUITE testbeds were the foundational environment for the fast failure detection experiment. This program used the Linux operating system as a base operating system and Linux R/K as a resource kernel module. The Ensemble group communication software system and additional network tools were also used in the QUITE testbed environment.

LINUX

The Linux[2] system was the base operating system used for the fast failure-detection program development and experiments. Stafford (2001) explains some overall details of the Linux system as follows: "The Linux kernel consists of several important parts: process management, memory management, hardware device drivers, file system drivers, network management, and various other bits and pieces…." Stafford illustrates several of these Linux® elements in his simplified structure breakdown of the Linux® kernel (Figure 3).

[2] Linux® is written and distributed under the GNU General Public License Version 2, June 1991.

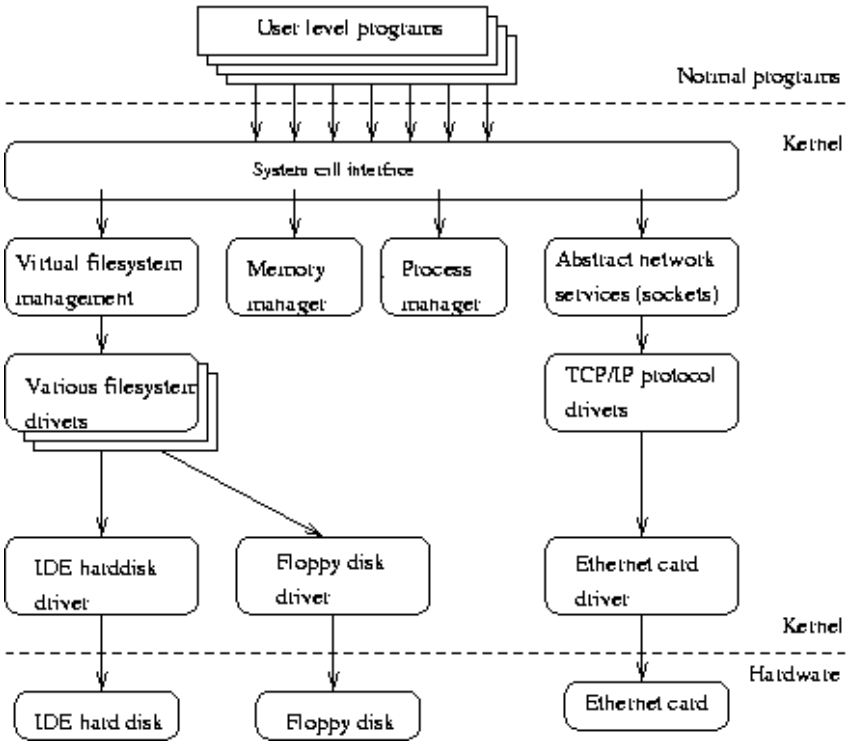

Figure 3. Linux® kernel.

The Linux® system device drivers can be found where the Linux® kernel borders the system hardware. Stafford (2001) describes mutual characteristics of these device drivers as follows: "At the lowest level, the kernel contains a hardware device driver for each kind of hardware it supports. Since the world is full of different kinds of hardware, the number of hardware device drivers is large. There are often many otherwise similar pieces of hardware that differ in how they are controlled by software. The similarities make it possible to have general classes of drivers that support similar operations; each member of the class has the same interface to the rest of the kernel but differs in what it needs to do to implement them. For example, all disk drivers look alike to the rest of the kernel, i.e., they all have operations like `initialize the drive', `read sector N', and `write sector N'. ”

LINUX/RK

The Linux/RK system has been used as the QoS resource manager for the fast failure detection program. CMU Real-time and Multimedia Systems Laboratory developed the Linux/RK system. *What is a Resource Kernel* (Carnegie Melon University, 1997) describes Linux/RK as follows: “Linux/RK stands for Linux/Resource Kernel, which incorporates real-time extensions to the Linux kernel to support the abstractions of a resource kernel. A resource kernel is a real-time kernel (operating system) that provides timely, guaranteed and enforced access to system resources for applications.”

Oikawa (1998) provides further information regarding the Linux/RK system and states that Linux/RK is basically an implementation of a resource kernel based upon the Linux® system. as illustrated by him in Figure 4. The Linux/RK system is an integration composed of a Linux kernel

and portable resource kernel subsystem. The resource kernel is a kernel module labeled the rk.o module, which is loaded into Linux® by following a sequence referenced in Komarinski (2000). The Linux/RK system used in the research discussed in this report based upon the Red Hat® Linux® version 6.2.

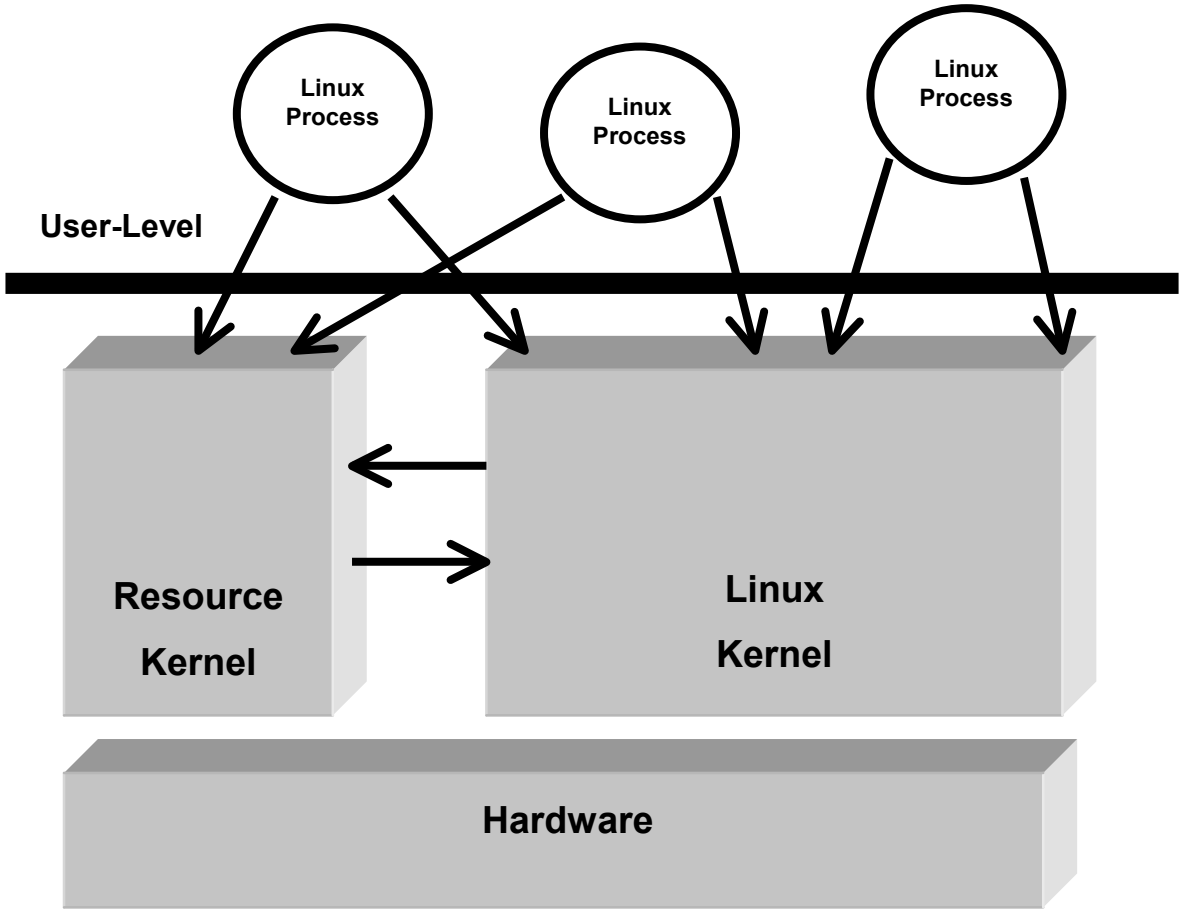

Figure 4. Linux/RK resource kernel architecture.

The Linux/RK system allows applications that require specific levels of QoS to use the resource kernel calls to acquire the needed resources. Oikawa (1998) describes this appropriation of resources as follows: "A QoS manager or an application itself can then optimize the system behavior by computing the best QoS obtained from the available resources."

ENSEMBLE

The communication system used in the failure detection research effort is Ensemble[3]. This system was developed at Cornell University Computer Science Department. The Ensemble system is a group communication software program that was constructed over the years by building upon other

[3] Ensemble was written and copyrighted in 1996 by Cornell University.

previously developed communication software, which was also developed at Cornell University. These other communication programs include Isis and Horus, with the Ensemble software essentially the third generation of these communication programs.

Birman (2000) explains the Ensemble architecture as follows: "The basic idea underlying both (Ensemble and Horus) projects is to support group communication using a single generic architectural framework within which the basic group communication interfaces are treated separately from their implementation." The composition of the Ensemble system includes many protocol layers as illustrated by Birman in figure 5, which shows some of these as micro-protocols. Birman continues with the Ensemble description by explaining the microprotocols: "One can then plug in an implementation matching the specific needs of the application. To maximize flexibility, each group end-point instantiates a stack of what we call micro-protocols. The developer arranges for the stack used in support of a given group to provide precisely the properties desired from the group. Each micro-protocol layer handles some small aspect of these guarantees. … Each process in a process group is supported by an underlying protocol stack; the stacks for the various members are identical, but the stacks used in different groups might be very different from one-another."

The typical use of Ensemble includes communication primitives and network tools as noted in Birman (1997): "Users of Ensemble treat it as a collection of tools and communication primitives."

For the purposes of this research, the Hot-C interface, Gossip server, Group Communication, heartbeat, and synchronization functionality of the Ensemble system are used within the fast failure-detection program.

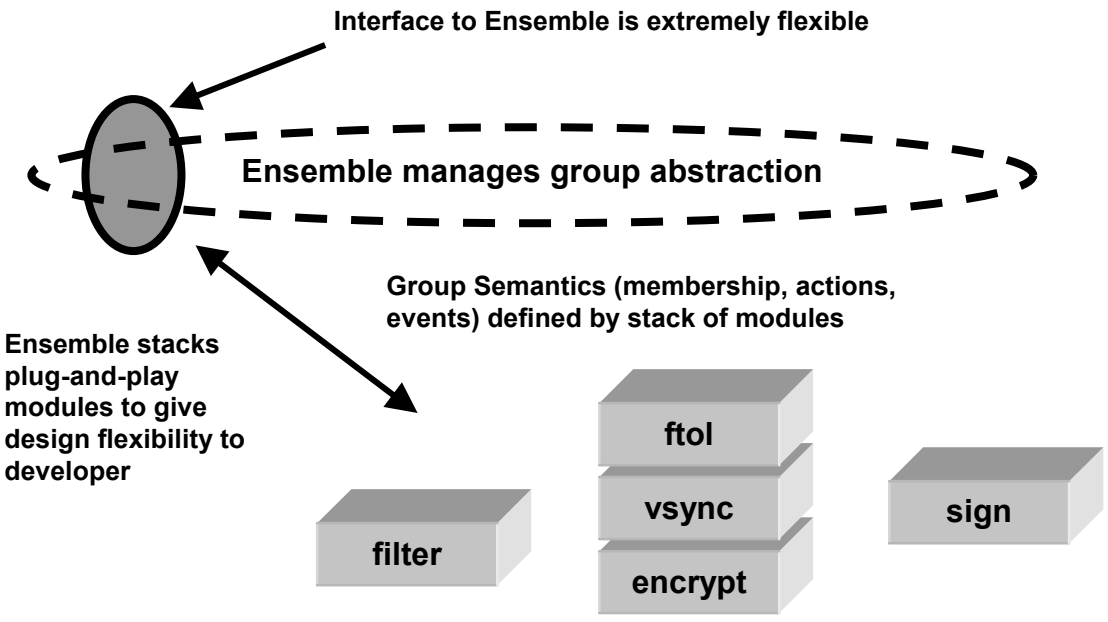

Figure 5. Layered microprotocols in Ensemble.

EXPERIMENT REQUIREMENTS

HARDWARE SETUP

The QUITE testbed used for the experiment was composed of COTS Intel®-based Pentium® architecture units with single processor systems that ran at 400 MHz and contained mostly generic components. The 100BaseT Ethernet network configuration provided communication for these systems. The network nodes were populated with 3Com® 100BaseT network devices and a typical high-density display system based upon the Intel740 video accelerator. The Microsoft Windows NT® version 4.0 build 1381 was the operating system implemented within this QUITE testbed. For simple monitoring purposes, this system included Service Pack 4. The primary operating system was Red Hat® Linux® version 6.2, which was used for experiment development, execution, and testing within the QUITE testbed. Standard generic drivers were used during these tests, and no modifications were performed upon these devices. The storage devices, primary and secondary, used on these systems were 512-MB SDRAM (60 ns) and 16-GB disk systems. The BIOS used by these systems was AwardBIOS™ version 4.51, with HAL: MPS 1.4-APIC platform. The internal clock mechanism was the WindowsNT® multimedia timer, which provided a clock resolution of < 1 ms for the Microsoft systems and the 8254 timer chipset for the Linux® systems.

SOFTWARE REQUIREMENTS

Linux/RK. The expected implementation of the fast failure detector was a separate process on CMU's resource kernel for Linux. Providing guaranteed CPU resources allowed it to generate and process the heartbeat messages more predictably. This experiment also used the C-language version of Ensemble.

Ensemble had to be modified to accept input about failed nodes or failed processes externally. Some mechanism had to be created for communicating between the separate fast failure detector process and the Ensemble communication protocol stack.

The distinctive fast-failure-detector (FFD) component developed used an initial algorithm that was simple. The generator sent messages to every other node at a rate about 2.5 times faster than the timeout period. The census taker asserted node failures if it had not received a message from any particular node for a period longer than the timeout period. The FFD component accepts dynamic modification of its execution parameters. However, the mechanism to effect the change was not crucial to the experiment and the design was left to the group implementing the component.

The fast failure detector was expected to be portable between normal Linux and Linux/RK execution environments. It was desirable, but not necessary, for the detector to also execute in the WIN32/WinSock environment.

Use of other Quorum components including Dynamic, Scalable, Dependable, Real-Time (DeSiDeRaTa), QoS Metrics Services (QMS), and Adaptive QoS Availability (AQuA) was not anticipated. However, it was anticipated that these components and component modifications would be used in other experiments and in the QUITE tool-kit. Coexistence with other components was considered in the design of components.

ANALYSIS PROCEDURE

HYPOTHESIS

The hypothesis of this experiment was that given a specified hardware configuration with a specified application load, use of the proposed fast failure detector based upon the CMU resource kernel (or OGI's QUASAR system) will allow an appropriate application to detect and recover more rapidly from hardware node failures. Specifically, such a system could be configured with a shorter heartbeat timeout period while maintaining an acceptably low level of false positives.

An auxiliary abstract hypothesis that was not investigated in this experiment should be considered during the project because it may represent the primary means for technology transfer. Does a separate failure detector function with guaranteed resources allow the user shorter timeout periods in practical systems? If this hypothesis is true, then separate, low-timeout failure detectors could be constructed using alternate facilities with dedicated access to resources such as in-kernel device drivers and outboard processors.

ASSUMPTIONS

The previous discussion highlights several assumptions about the criticality of the failure detection functional operation toward the reduction of the timeout period. These assumptions were investigated as part of the execution of the experiment.

The experiment assumed, of necessity, that the Quorum components could perform as expected. It was assumed that the performance of network message delivery throughout the entire Transmission Control Protocol (TCP)/Internet Protocol (IP) stack could be controlled through reserved resources within the resource kernel.

The isolation of the heartbeat function slightly changed the failure semantics. In the normal Ensemble system, receipt of a heartbeat message indicates that the Ensemble component on the remote node is (at least partially) functional and believes that other components on the same node are still operational. In the FFD configuration, receipt of a heartbeat message showed only that the FFD component on the remote node was still operational. The primary purpose of the FFD was to detect node failures, which would lead to failures of the FFD, the Ensemble component, and all other processes on the affected node. The standard Ensemble failure detector had to remain enabled to detect failures of the Ensemble and application components.

Finally, the applicability of the results of this experiment to NSWC's HiPer-D testbed were predicated on the assumption that the variability in message delivery times in the AAW path was primarily due to anomalies in CPU resource management. This assumption could not be tested as part of the QUITE experiment.

DATA METRICS

The QUITE experiment had two phases. The first operated the FFD as a single component, varying the period between heartbeat messages and the selected timeout period and investigating the effect of various background processing loads. The resulting false positive rates were graphed onto a set of curves that were similar to the “message delivery time” distribution curve described above. These measurements produced no outstanding hypothesis.

The second phase of the experiment compared the behavior of the normal Ensemble-based system with the FFD-enhanced system. We experimentally determined an appropriate background CPU load for the application execution.

Note that the relevant metric in both cases was the timeout period that was assigned to the heartbeat function. The time for an application to recover from a node failure was a more direct metric of the relevant property. The time to execute the relevant test cycle (execute test, fail a node, reboot the node, reestablish the application environment) was over two orders of magnitude slower than the transmission of a single message and appeared impractical in the current environment. Each test cycle would also likely require human intervention. The relevance of the timeout period metric to the actual application-level metric was validated through spot checks during experiment execution.

ANALYSIS TECHNIQUES

The detailed experiment was based upon the performance of message delivery using the resource kernel. Depending on the variability of the results, the system performance had to be analyzed under various application loads. It was not whether the performance difference between the control and experimental system configurations should be compared by absolute differences, as ratios, or in some other way. The primary criterion for comparison was that the results be statistically significant. Specific real-time data analysis was not required to deliver meaningful results. Some visualization of run-time performance assisted in the conduct of the experiment.

FAST FAILURE DETECTION PROGRAM

The FFD could be built and executed on its own or it could be executed while taking advantage of facilities like Linux/RK and Ensemble group communication. By default, the FFD was constructed using Linux/RK facilities and Ensemble.

The FFD program used the group communication protocol stack to factor out real-time failure detection elements (Figure 6). The Census Taker thread used system resources through initialization and direct requests to the Linux resource kernel “Linux/RK.” The heartbeat thread also performed initialization and executed direct resource request calls into the Linux resource kernel. This program then executed the failure detection function within the operational Linux resource kernel environment. The FFD program was developed based upon real-time programming techniques and used reserved CPU resources. The FFD uses a heartbeat with deadline function to reliably detect host failures in sub-second time frames, even in the presence of the competing CPU loads.

The FFD notified applications about node failures and provided regular reports to the resource manager on host status. The FFD could also provide more metrics, such as whether a host was in danger of missing its heartbeat deadline, which would cause a "false positive" failure indication. However, the FFD does not detect group member failures. This group member failure detection would require that the special real-time programming techniques be applied to the overall application. Instead, each FFD actually detected failure of other FFD components on other nodes. The dependency tree information mentioned earlier allowed us to reason about the effect on the overall system. FFD failure was highly correlated to failure of the node upon which it operated. The mission-critical application was extensively reviewed and tested so that the most likely cause of its failure was due to the failure of the underlying node, which in the most relevant context was most likely due to battle damage, the intrusion of an foreign object into the host hardware.

Finally, we retained the original failure detection capability of the group communication system, which continued to detect failures of group members—now due to less likely causes and with a much lower probability of occurrence.

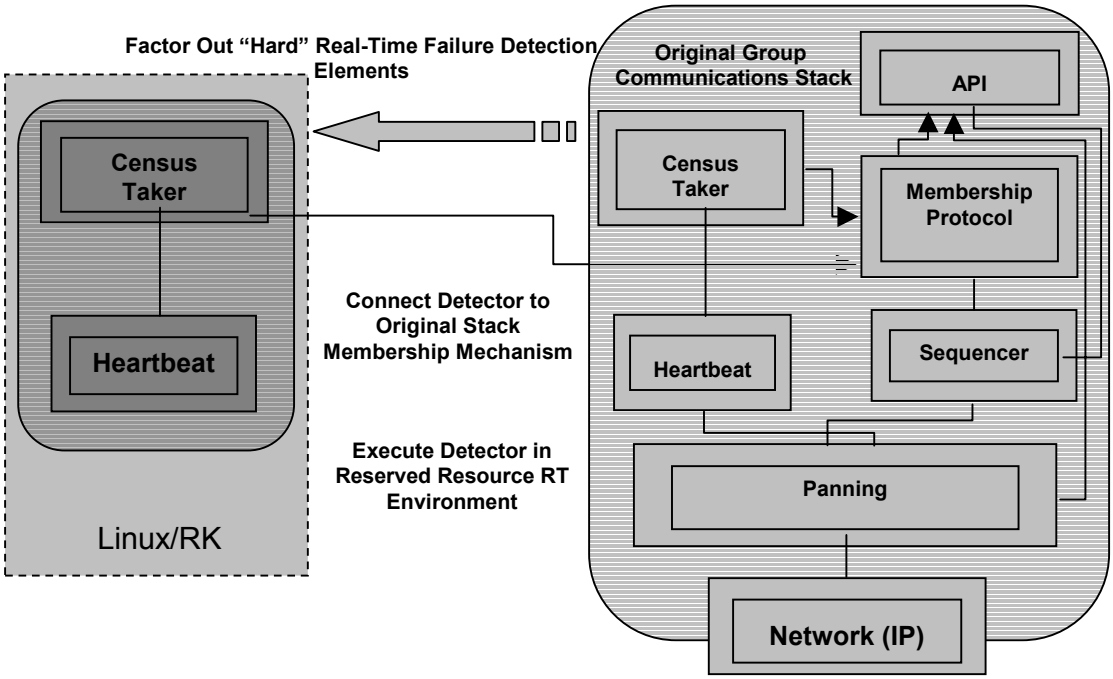

Figure 6. FFD program.

CONCLUSION

EXPERIMENT RESULTS

Figure 7 shows the notional false alarm results from the FFD experiment. The overall experiment results showed that the FFD program detects node failure within a sub-second timeframe.

The initial investigation, including the FFD experiment, supported our belief that knowledge about faults can be effectively incorporated into resource allocation decisions and that this information can improve the coordination between applications relative to resource sharing. The open group is building a real-time group communication product that uses the FFD, and they hope to extend the concepts with further research in applying hierarchical resource management in large, complex systems built using Common Object Request Broker Architecture (CORBA) and Advanced Computing Environment (ACE™)/Telephony Application Object (TAO™) technology.

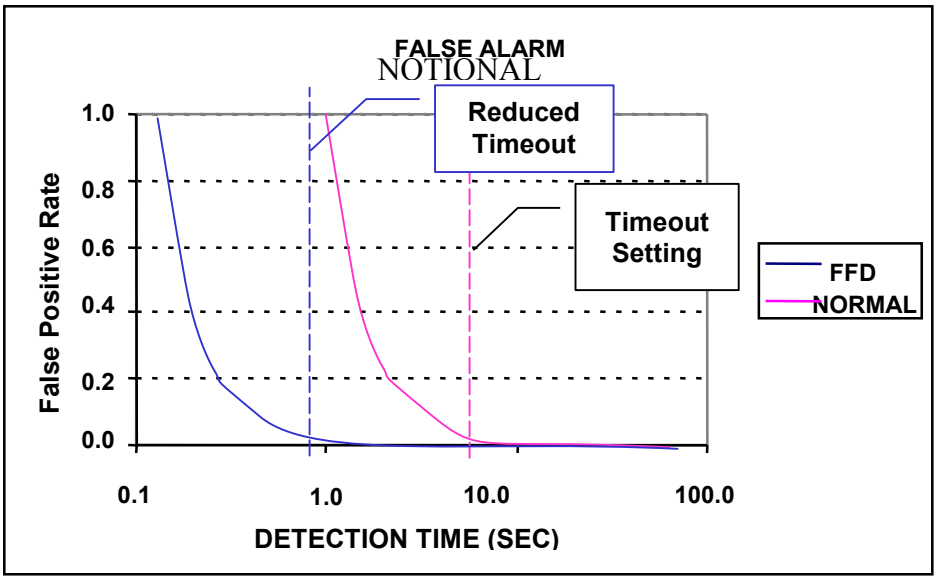

Figure 7. False positive rate.

REFERENCES

Birman, K., W. Vogels, K. Guo, M. Hayden, T. Hickey, R. Friedman, S. Maffeis, R. VanRenesse, and A. Vaysburd, A. 1997. "Moving the Ensemble Groupware System to Windows NT and Wolfpac*k,*" *Proceedings of USENIX Windows NT Workshop*, 11–13 August, Seattle, WA.

Birman, K., et al. 2000. "The Horus and Ensemble Projects: Accomplishments and Limitations," *Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX '00),* January. Hilton Head, SC.

Carnegie Mellon University. 1997. *What is a Resource Kernel*, Computer Science Department, Real-Time and Multimedia Laboratory Web-site, http://www-2.cs.cmu.edu/afs/cs/project/art-6/www/resource-kernel.html, October, 14, 1997

Kmoarinski, M. and C. Collett. 2000. *Red Hat Linux System Administration Handbook*. Prentice Hall, Inc. Upper Saddle River, NJ.

Oikawa, S. and R. Rajkumar. 1998. *Linux/RK: A Portable Resource Kernel in Linux*, IEEE Real-Time Systems Symposium Work-In-Progress (Dec). Madrid, Spain.

Stafford, S., L. Wirzenius, and J. Oja. 2001. *The Linux System Administrator's Guide, Version 0.7*, http://www.linuxdoc.org/LDP/sag/.

APPENDIX A
FFD CONFIGURATION AND INSTALLATION

The Fast Failure Detector (FFD) software requires prerequisite software before it can be built and executed. This appendix briefly describes the process of building the prerequisite software. For more detailed information, URLs to the various vendor sites are also given.

This distribution consists of the following installation tar/zip files:

Ffd_src.tar FFD C++ source code and makefiles

Ggc_src.tar GGC C++ source code and makefiles

Rk.tgz Zip file containing the Linux RK distribution

Ocaml-3.00.tar.gz Gzipped file containing the Ocaml-3.0 distribution

Ensemble-1.00.tar.gz Gzipped file containing the Ensemble 1.0 distribution

The following documents are also in this distribution

FFD_Experiment.htm HTML version of the experiment2 description

Fault_management.htm HTML version of an overview white paper on FFD

README_FIRST.htm This file (an overview of the software)

README.txt A textfile containing instructions on building and running the GGC and FFD software

The required software is the following (these should be installed in the following order):

1. Linux 6.2
2. Linux RK
3. Ocaml 3.0
4. Ensemble 1.0

In the following instructions, allow $BUILD_PATH to be wherever the software is going to be built. Allow $DOWNLOAD_PATH to be wherever the install tar files will be stored.

How to build Linux RK

Un-tar rk.tgz file

cd $BUILD_PATH

tar zxvf $DOWNLOAD_PATH /rk.tgz

1. cd LinuxRK
2. cd linux.2.2.14.rk
3. make menuconfig
4. make dep
5. make clean
6. make bzImage

6b. do a make at the top level directory

7. make modules
8. su

   make modules_install

9. copy the bzImage to /boot:

   under dir $BUILD_PATH/LinuxRK/linux.2.2.14.rk/arch/i386/boot

   cp bzImage /boot/linux-2.2.14-rk

10. copy the System.map to /boot

    under dir $BUILD_PATH/LinuxRK/linux.2.2.14.rk

    cp System.map /boot/System.map-2.2.14-rk

11. edit /etc/lilo.conf to add the linux-2.2.14-rk kernel

    image=/boot/linux-2.2.14-rk

      label=linux-rk

      read-only

      root=/dev/hda1

12. /sbin/lilo

13. reboot

at lilo boot: prompt type in linux-rk

How to build ocalm 3.0

Un-tar ocaml-3.00.tar.gz file

cd $BUILD_PATH

tar zxvf $DOWNLOAD_PATH /ocaml-3.00.tar.gz

1. cd $BUILD_PATH/ocaml-3.00

open install file and follow the instructions (skip option 5 & 8)

The instructions say the following:

./configure

make world

make bootstrap

make opt

su

umask 022

make install

make clean

How to build ensemble

---

un-tar ensemble-1_00.tar.gz

cd $BUILD_PATH

tar zxvf $DOWNLOAD_PATH /ensemble-1_00.tar.gz

1. cd $BUILD_PATH/ensemble

2. Make sure that the MACHTYPE,OSTYPE,CAMLLIB environment variables are set correctly before building.

export MACHTYPE=i386

export OSTYPE=linux

export CAMLLIB=/usr/local/lib/ocaml

open install file and follow the instructions (need to use web browser to open INSTALL.html)

3. cd $BUILD_PATH/ensemble/def

   type --> make clean

   type --> make

4. create HOT C interface

   cd $BUILD_PATH/ensemble/def

   type --> make libhot

Please refer to the "Readme.txt" document for a detailed discussion on building the GGC and the FFD. Included here is a brief quick installation guide for building these two pieces of software.

How to build GGC (Group Communication)

---

cd $BUILD_PATH

tar xvf ../downloads/ggc_src.tar

1. cd ggc

   vi Makefile

   change line --> ENS_INC_DIR=$BUILD_PATH/ensemble/hot/include

   change line --> ENS_LIB_DIR=$BUILD_PATH/ensemble/def/obj/i386-linux

   change line --> LINUXRK_ROOT=$BUILD_PATH/LinuxRK

2. make

How to build FFD

---

cd $BUILD_PATH

tar xvf ../downloads/ffd_src.tar

1. cd ffd

   vi Makefile

   change line --> ENS_INC_DIR=$BUILD_PATH/ensemble/hot/include

change line --> ENS_LIB_DIR=$BUILD_PATH/ensemble/def/obj/i386-linux

change line --> LINUXRK=$BUILD_PATH/LinuxRK

2. make

EXPERIMENT INSTRUCTIONS

As mentioned earlier, the FFD can be built and executed on its own or it can be executed while taking advantage of facilities like Linux/RK and Ensemble group communication. By default, FFD will be built using Linux/RK facilities and Ensemble. To build FFD using these facilities, the ./ggc directory must be built first, followed by the ./ffd directory.

Platform

The primary platform for FFD with Ensemble is on Linux. FFD without

Ensemble has been built on Solaris. ./ggc/ggc.c requires minor

changes related to pthread_once() call on Solaris.

Common Makefile Variables

Set ENS_INC_DIR and ENS_LIB_DIR to point to the directory where

Ensemble HOT C interface include files and libhot.a can be found

respectively. BINDIR controls where the final binary executables will

be generated. Note that the current version will not remove the

binaries when executing 'make clobber' unless BINDIR is set to ".".

Building ggc

GGC is a Generic Group Communications library which makes it easier to

access packages like Ensemble. By default, Ensemble is used as the

underlying group communications package.

% cd <ggc_dir> (i.e. cd ./ggc)

% make

This will make the library libggc.so and a control program "ggccp."

Building FFD

Set GGC_INC_DIR and GGC_LIB_DIR to point to the directory where

Generic Group Communication (GGC) interface include files and libggc.so

can be found respectively. By default, they are set to ../ggc.

Set ENS_INC_DIR and ENS_LIB_DIR to point to the directory where

Ensemble HOT C interface include files and libhot.a can be found

respectively.

Many compile time options that can be toggled. The

following describes the make variables that should be commented or

uncommented as desired.

The following GGC compile time options are enabled by default:

USE_ENS_GROUPD=-DUSE_ENS_GROUPD - enabling it causes FFD to specify

that the Ensemble group daemon should be used. Do NOT set this

variable when building FFD t run the QUITE experiment (instructions

for running the experiment is elsewhere in this document).

USE_GGC_BCAST_SUSPECT=-DUSE_GGC_BCAST_SUSPECT - use GGC to broadcast

that a node has been declared down. The message is of the form

<1:IP_address:time.stamp>

The following GGC related compile time options are turned off by default:

USE_GGC_SUSPECT=-DUSE_GGC_SUSPECT - mark all endpoints on a downed node using the hot_ens_Suspect() call when remaining FFDs declare of a downed node.

FFD_HB_TO_GGC_MSG=-DFFD_HB_TO_GGC_MSG - replicate the FFD heartbeat messages and send it out using GGC broadcast mechanism. Enabling this option is not recommended.

GROUP_LIST_DELIM=-DGROUP_LIST_DELIM=':' - specify the character to be used to separate multiple groups.

FFD_ENS_DEBUG=-DFFD_ENS_DEBUG - force FFD to generate GGC debug information even if FFD debug is not enabled.

USE_FFD_REPORTER=-DUSE_FFD_REPORTER - enables the external interface via which other entities can receive host down messages without using multicast or GGC. The message is written out in XML format (DTD in ./ffd/ffd.dtd) to a named pipe/FIFO, /tmp/ffd_fifo (see FFD_REPORTER_FIFO defined in ./ffd/utils.h).

FFD can take advantage of Linux/RK when compiled and run on a Linux/RK-based kernel. Linux/RK based kernel is automatically detected if current kernel running on the system is named *-rk. Set LinuxRK to point to the top level directory of the Linux/RK tree (i.e., /LinuxRK) such that RKKERNEL and RK_LIBS libs are set to point to the appropriate sub-directories below in the

source tree. To force the build to use RK facilities on a system which do not have a Linux/RK kernel named as *-rk, set OSTYPE to "LinuxRK" in the Makefile.

% cd <ffd_dir> (i.e. cd ./ffd)

% cd make

This will make the following five executables:

- ffd
- hbeatstats
- runcensustaker
- runheartbeat
- setparams

By default, the above programs will communicate over basic multicast. An additional program, iterator, is also built. iterator is a simple looping program to load the CPU; it can also be built to use Linux/RK facilities.

You may need to set the LD_LIBRARY_PATH path to include the directory where libhot.a and libggc.so is located.

Running FFD

Ensure that at least one Ensemble gossip server is running. You may need a gossip server on each participating node when the "-also_groupd" option is specified. The following Ensemble related environment variables also must be set:

ENS_GROUPD_PORT=2999

ENS_GROUPD_HOSTS=quite02_t:quite03_t:quite04_t:quite05_t:testbed01:testbed02:testbed03:testbed04

ENS_GOSSIP_PORT=1998

ENS_PORT=1999

ENS_GOSSIP_HOSTS=quite02_t:quite03_t:quite04_t:quite05_t:testbed01:testbed02:testbed03:testbed04

Optional Ensemble environment variables:

ENS_HEARTBEAT=20

ENS_MAESTRO_PARAMS=suspect_max_idle=12:int;suspect_sweep=1.000:time

Note that the names of hosts where gossip servers run are specified in ENS_GOSSIP_HOSTS and ENS_GROUPD_HOSTS; each host name is separated ":" as illustrated in the example above.

Execute "ffd -u -v -G" on the nodes where FFD is to be run. The "-G" flag causes FFD to use GGC and joins a group named ffd_<simple_host_name> where simple_host_name> is the unqualified name of the host that is running "ffd." When a node goes down, a downed node status will be broadcasted to all members in that group via GGC. Note that FFD will not broadcast when a node comes up or the status of each node periodically using GGC; only node down status is re-broadcasted using GGC. This message can be monitored by an program which joins the group ffd_<simple_host_name>. In the absence of application programs, use "./ggc -M -G -g ffd_<simple_host_name>" from the GGC directory to validate FFD messages. The "-G" for the "ggc" command directs it to use Ensemble groupd (the group daemon).

Group name can be overridden by using the "-g" option. Multiple group names can be specified by ":" (or whatever was specified as the delimiter in GROUP_LIST_DELIM during the build) when using the "-g" option.

The "-u" flag asserts urgency and uses the Linux/RK reservation facilities on systems with Linux/RK kernel.

FFD heartbeats can be monitored by running the command "hbeatstats." By default, it will monitor heartbeats sent using multicast.

To test the various communications infrastructures, execute "runheartbeat -G -S" on each node to generate the basic FFD heartbeats, and run "hbeatstats -S" or "hbeatstats -G" on each node.

Note that the "-S' and the "-G" flags should not be used simultaneously on the same command line as it will receive duplicate heartbeat messages from each node. Heartbeats broadcasted via GGC on any node can be monitored from any machine by running "hbeatstats -G -g ffd_<simple_host_name>".

Notes

FFD is designed to be kept running continuously; however, there is cleanup code which should get executed upon exit. On Linux/RK based systems, asserting urgency via the "-u" option to use RK facilities may not always cleanup properly. It has been observed that FFD will not exit until the RK resource set created by FFD has been removed (via rkdestroyRS <ffd_resource_set_ID>. On systems when FFD has been brought up and down many times, there may be zombie FFD processes left in the system. They do not appear to cause any problems on the surface. Another rare problem that has been observed is that the RK machine may freeze up after running various experiments for sometimes. A reboot is usually necessary to bring the system back to normalcy. It is not clear whether the problems are due to Linux/RK, FFD's use of RK facilities, or a problem with kernel configuration during the kernel build.

QUITE Experiment with FFD and Ensemble

This section describes the necessary steps to only setup the machines to run the FFD experiments.

A minimum of three machines, preferably running RK, is recommended for running the FFD experiment. An additional machine should be used for monitoring and collecting the data; this machine need not run RK but does need to have Ensemble configured.

FFD should be built without enabling use of the Ensemble group daemon (i.e. USE_ENS_GROUPD should be commented out in ./ffd/Makefile). Also note that the USE_ENS_GROUPD flag should not be turned; non deterministic behavior has been observed with hot_ens_Suspect() when groupd is in use.

The following should be performed on FFD machines following reboot:

1. Synchronize the system clocks on all machines

2. Load the Linux/RK kernel module if it is not loaded by default (you have to be user "root")
insmod ./LinuxRK/rkmod/rk.o

3. Start a gossip server on only one of the machine. This machine will be designated as the "leader" % gossip &

The network will NOT be disconnected on this machine.

4. Start FFD

% ./ffd/ffd -v -u -G -g ffd

5. Start the heartbeat generator

% ./ffd/runheartbeat -G -g ffd

On a fourth machine, run the following:

1. Start the GGC control program which has been modified to monitor

the host down message from FFD and to issue the hot_ens_Suspect()

call to mark all group members known to be on the host that was

declared down by FFD.

% ./ggc/ggccp -M -g ffd -D 3

"-D 3" specifies debug level 3; otherwise, only the IP address of the downed host will be displayed when a hot_ens_Suspect() is received. "ggccp -M" will prompt for a message to be entered which is sent to all group members. It is ignored by the members participating in the experiment unless a host down message sent by the FFD is entered (not recommended).

2. Start the monitoring program to receive heartbeats sent by

runheartbeat using Ensemble

./ffd/hbeatstats -G -g ffd

The output should be piped to a file for collecting the timing information to be analyzed later.

3. Start the monitoring program to receive heartbeats sent by FFD over multicast

./ffd/hbeatstats -S

This is purely for monitoring network connectivity. The output should be piped to a file for collecting the timing information to be analyzed later.

The machines are now ready for conducting the experiment. One of the simple methods for bringing a node "down" is to disconnect it from the network. This is easily achieved by disconnecting the network cable on each machine not designated as the "leader." Reconnecting it back in will bring the host back up as far as FFD is concerned.

INITIAL DISTRIBUTION

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (4)

SSC San Diego Liaison Office
C/O PEO-SCS
Arlington, VA 22202–4804

Center for Naval Analyses
Alexandria, VA 22311–1850

Office of Naval Research
ATTN: NARDIC (Code 362)
Arlington, VA 22217–5660

Government-Industry Data Exchange
Program Operations Center
Corona, CA 91718–8000

Defense Advanced Research Projects Agency
Information Technology Office
Arlington, VA 22203–1714

Naval Postgraduate School
Monterey, CA 93943–5101

Teknowledge
Palo Alto, CA 94303 (3)

The Open Group
Woburn, MA 01801 (3)

System/Technology Development Corporation
Herndon, VA 20170–4214 (3)